Инициализация переменных

В языке Си, равно как и во многих других языках высокого уровня, можно определять значение переменной сразу в момент ее объявления. Например, так:

```language-c
int total = 1024;
```

Визуально это выглядит как обычное присваивание переменной значения. Но в действительности – это совершенно другая операция, которая называется **инициализацией** переменной. То есть, когда мы объявляем какую-либо переменную (любого типа) и сразу указываем для нее некоторые начальные данные, то в этот момент запускается механизм инициализации этой переменной. Как будет показано в будущем, при инициализации возможны конструкции, которые нельзя использовать при присваивании, а пока нужно запомнить, что в большинстве случаев присванивание и инициализация - это разные вещи.

Для полноты картины приведу еще один пример, когда мы комбинируем обычное объявление и инициализацию переменных:

```language-cpp
int total = 1024, buffer;
```

Синтаксически так тоже можно делать и использовать в своих программах.

Целочисленные и символьные литералы

Теперь давайте внимательно посмотрим на 1024 в нашей программе. И зададимся вопросом, как его воспринимает компилятор и где оно хранится? В программировании явно прописанные числовые значения называются **числовыми литералами** и представляются как целочисленные константы типа int (Так задумал создатнль языка - Денис Ритчи: все константы, *записанные в программе в десятичном виде*, хранить на уровне типа int). Важно уточнить, что правило работает только да того момента, пока число помещается в диапозон возможных значений. Если целочисленный литерал положителен и не умещается в этот диапазон, то компилятор подбирает соответствующий размер типов данных в порядке возрастания:

int, unsigned int, long, unsigned long, long long, unsigned long long

Если же и *самого большого недостаточно* (что сложно себе представить в реальных задачах), то компилятор выдаст *ошибку*.

Но так только с целыми числами записанными в десятичной форме. Язык Си позволяет определять в программе числовые литералы в шестнадцатеричной и восьмиричной формах. Пример:

```cpp
int dec, hex, oct;
dec = 100;
hex = 0x1FA;
oct = 0123;
```

language-cpp

То есть, для записи шестнадцатеричных чисел перед ними ставится префикс в виде символов «0x», а для записи восьмеричных – префикс в виде нуля. Так можно прописывать любые числа в нужном нам формате. Разумеется, на уровне машинных кодов они представляются единым образом в виде набора бит и запись числовых литералов в той или иной форме служит исключительно для удобства восприятия программистом. Не более того.

Но с шестнадцатеричными и восьмеричными литералами есть один важный нюанс. Их компилятор изначально представляет не типом int, как десятичные, а типом **unsigned int**. Соответственно, если литерал не умещается в этот тип, то берутся другие больших размеров в порядке:

unsigned int, unsigned long, unsigned long long

При желании мы можем явно указать компилятору тип числового литерала. Для этого используются следующие суффиксы:

- U или u – использование модификатора unsigned в определении литерала;
- L или l – использование типа long при определении литерала;
- LL или ll – использование типа long long при определении литерала.

Хорошо, с запись литералов разобрались. Но как они сохраняются непосредственно в программе? Конечно же, компилятор переводит их в машинный код и хранит непосредственно в скомпилированной программе как неизменяемые данные, то есть, как константы. Когда программа загружается в память компьютера, то вместе с ней загружаются все данные, которые мы явно определяем в тексте.

Отлично, с этим разобрались. Давайте теперь посмотрим на тип *char*, который формально определен и как символьный и как целочисленный.

Первый вопрос, как такое может быть? На самом деле все очень просто. Объявим переменную этого типа, например:

```language-cpp
char ch
ch = 'd';
```

Запись символа происходит в одинарных кавычках, и только так можно записывать *символьные литералы* в языке Си. Когда компилятор видит одинарные кавычки, то он воспринимает информацию в них, как символ. Никакие другие кавычки для этого использовать нельзя. Например, двойные зарезервированы для определения строк, а запись без них приведёт к тому, что d будет восприниматься как переменная.

Хорошо, на уровне программы мы теперь знаем, как прописывать отдельные символы. Но спрашивается, как эти символы представляются в машинных кодах? Там же могут быть *только числа*?

Все верно. Все символьные литералы в программе переводятся в соответсвующие коды. Например, в кодировке ASCII символу d соответсвует код 100. *Важно помнить, что ASCII - не единственная таблица кодов*, поэтому при замене самого символа на его код из таблицы и компиляции программы с использованием другой кодировки, *сам символ может поменяться* (например, стать не d, а u).

Далее, мы можем вывести это значение переменной ch с помощью функции printf() в двух форматах: символьном и числовом:

```language-cpp
printf("ch = %c, code = %d\n", ch, ch);
```

Об этой функции мы еще подробнее будем говорить, пока только отмечу, что вместо символов «%c» будет подставлено значение ch и переведено в символ. А вместо «%d» будет также подставлено также значение из ch, но выведено в виде целого десятичного числа.

После запуска программы увидим:

ch = d, code = 100

Этот факт показывает, что компьютеру важно лишь какое число хранится в переменной ch, а его интерпретация может быть самой разной: или как символ, или как целое число. Причем, обратите внимание, компилятор преобразовывает символьные литералы в числа типа int, а не char, как можно было бы ожидать. Это, как раз связано с тем, что он воспринимает

любой символ, как десятичное число, а оно по умолчанию представляется типом int.

Вещественные литералы

Помимо целочисленных в программе можно прописывать и вещественные литералы. Определять их можно следующими способами:

```language-cpp
double d1, d2, d3, d4;
d1 = 10.0;
d2 = -7.;
d3 = 1e2;
d4 = 5e-3;
```

В первых двух случаях используется символ точки. Причем, обратите внимание, несмотря на то, что числа 10.0 и -7.0 с математической точки зрения соответствуют целым числам 10 и -7, *на уровне программы они будут вещественными и иметь тип double.*

Все вещественные литералы компилятор языка Си по умолчанию представляет этим типом данных. Соответственно, математические операции с числами 10.0 и -7.0 будут выполняться несколько иначе, чем с аналогичными целыми числами. Это следует иметь в виду.

Последние два варианта – **запись числа в экспоненциальной форме**:

<число>e<степень десятки>

Например:
1e2 = 1 *10 ^ 2 = 100*
5e-3 = 5 10 ^ -3 = 0,005

При желании мы можем явно указать компилятору переводить вещественный литерал в тип float. Для этого после числа следует прописать суффикс f, например, так:

```language-cpp
d1 = 10.0f;
```

В такой записи вещественное *число 10.0 будет представляться типом float, а не double*. Это бывает полезно, когда используется переменная типа float и ей правильно было бы присвоить значение того же типа:

```cpp
float var_f;
var_f = 10.0f;
```

Тогда компилятор не выдаст предупреждение (warning) о возможной потере данных в момент присваивания значения переменной var_f.

Операция sizeof

В конце отмечу довольно распространенную операцию sizeof языка Си. **Она возвращает число байт**, занимаемых в памяти переменной или, отведенных под тип данных. Синтаксис этой операции следующий:

sizeof(<тип | имя переменной>);
sizeof <имя переменной>;

Обратите внимание, во втором случае мы можем записать ключевое слово sizeof без круглых скобок, но тогда эта операция *применяется только к переменным, но не к типам.* Чтобы не запоминать эти тонкости, обычно sizeof записывают с круглыми скобками и указывают либо тип данных, либо имя переменных. Например:

```cpp
int size_float = sizeof(float);
int size_var_f = sizeof(var_f);
```

На выходе получаем число байт, которое занимает тип float и переменная var_f.